# Trusted Computing using Enhanced Manycore Architectures with Cryptoprocessors

C. Mancillas López[1], M. Méndez Real[2], L. Bossuet[1], G. Gogniat[2], V. Fischer[1], A. Baganne[2]

[1]Laboratoire Hubert Curien
University of Lyon
42000 Saint-Etienne, France
cuauhtemoc.mancillas.lopez@univ-st-etienne.fr

[2]Lab-STICC
University of Bretagne-Sud
56321 Lorient, France
maria.mendez@univ-ubs.fr

***Abstract*—Manycore architectures correspond to a main evolution of computing systems due to their high processing power. Many applications can be executed in parallel which provides users with a very efficient technology. Cloud computing is one of the many domains where manycore architectures will play a major role. Thus, building secure manycore architectures is a critical issue. However a trusted platform based on manycore architectures is not available yet. In this paper we discuss the main challenges and some possible solutions to enhance manycore architectures with cryptoprocessor.**

***Keywords—manycore architectures, symmetric cryptography, key exchange, cryptoprocessor***

## I. INTRODUCTION

Secure handling of personal data and privacy in manycore architectures is a major issue. The economic and social challenges are numerous as this type of architecture will be massively deployed in the future both in infrastructure such as "cloud computing" and in most embedded systems constrained in resources and performance. It is thus essential to address the question of the definition of these architectures in terms of not only performance but also security to ensure a large adoption of these technologies by end users. Lack of trust will be a hindrance to economic development, the challenges are immense.

To propose an efficient and secure solution it is necessary to enhance hardware manycore architectures with closely coupling of heterogeneous processing resources (some dedicated to the processing of data in clear and some dedicated for treatment of protected data). It is also necessary to rethink the relationship between software and hardware to ensure a protection in depth. Today these issues are too often neglected resulting in solutions developed at the end of the design cycle as an afterthought. It is essential to provide a breakthrough in these design approaches to provide trusted manycore architectures by building hardware and software.

This paper proposes an analysis of trust building to execute independent applications in parallel, securely and ensuring respect for the privacy of users. For this, several points are discussed: 1) proposition of a processing cluster to run both algorithms for processing information and cryptographic algorithms (with a strong level of coupling for performance reasons while ensuring no leakage of information), and 2) proposition of a manycore architecture integrating heterogeneous clusters for secure cryptographic. One key point discussed in this paper is related to the use of several cryptoprocessors for a single application. This point needs a deep analysis, especially for key exchange in order to guarantee no leakage of information.

The paper is organized as following. Section II discusses some challenges to enhance manycore architectures with cryptoprocessors. Section III illustrates how these challenges can be handled using an existing manycore solution named TSAR. Section IV provides an analysis of points that still need to be addressed and Section V concludes the paper.

## II. CHALLENGES TO ENHANCE MANYCORE ARCHITECTURES WITH CRYPTOPROCESSORS

Manycore based systems will become the mainstream in a near future, it is thus mandatory to think these architectures with security properties by construction. Cloud computing is one of the numerous application domains that will rely on this technology. Several applications can be envisioned, for example secure access to private information. End users will perform many requests in parallel in order to retrieve, to search for, and to classify some personal data (e.g. text, images, and video). It is thus necessary to combine both processing power and cryptographic functions. In order to build a solution to execute many independent applications in parallel in a secure way, several points need to be targeted: 1) building of a processing cluster enhanced with cryptographic resources, 2) building of a manycore architecture integrating heterogeneous clusters for secure cryptographic, 3) building of mechanisms for logical isolation (in software) and physical (hardware level) to ensure execution of partitioned applications, 4) joint building of software layers (driver, API ...) and hardware to provide a chain of trust and 5) proposition of strategies for dynamically distributing applications on a manycore architecture taking into account the security needs. All these contributions need to be addressed to build an efficient and

trusted execution platform. In this paper points 1) and 2) are discussed.

Using cryptoprocessors inside manycore architectures raises many challenges related to software and hardware layers. Links with the operating system and architecture of cryptoprocessors correspond to key issues. In this section some of these challenges such as heterogeneous clusters building, Trusted Computing Base analysis, and parallelism exploitation to improve the computation of cryptographic algorithms and the key exchange between cryptoprocessors are discussed.

### *A. Heterogeneous clusters building*

The idea of enhancing GPPs using some coprocessor exclusively dedicated to perform some tasks that require intensive computation, such as numeric calculus or cryptography is not new. The use of heterogeneous clusters has been widely studied using for example Graphics Processing Units (GPUs) to perform intensive computations and thus reducing the GPP workload [4].

Security is a very important service that a cluster in manycore architecture should hold. However cryptographic algorithms are based on operations that are generally not efficiently computed in GPPs such as bitwise operations, modular arithmetic, Galois field arithmetic, etc. [1]. So adding cryptoprocessor within a cluster will improve the performance of the cluster. Indeed, cryptographic algorithms implemented in software take thousands of clock cycles [2], while using a dedicated hardware coprocessor reduces the execution time to just few tens of clock cycles [3]. So if the GPP delegates the cryptographic computations to the cryptoprocessor, its workload is significantly decreased. The saved time can be used to perform other tasks as image or signal computation.

The goal of the cryptoprocessor is to perform all cryptographic computations and to provide a secure generation, management and storage for session keys. When the amount of data to be processed by the cryptoprocessor is significantly large, several cryptoprocessors can be used in parallel to accelerate the process. Indeed several execution contexts can be considered. 1) An application requires a single cryptoprocessor. In that case the application will delegate the cryptographic computation to the cryptoprocessor. The key generation will be performed by the cryptoprocessor itself and no leakage of the key can occur. 2) An application requires several cryptoprocessors to handle a large amount of data while respecting a required bandwidth. In that case, data will be split into several sets and each set will be delegated to one cryptoprocessor. In that case the key management step is more complex as all cryptoprocessors need to share the same session key. A dedicated key exchange protocol within the architecture needs to be built. Key leakage needs to be considered with lot of care. 3) Several applications require a single cryptoprocessor. That case is similar to case 1) when preemption is not authorized. Such an assumption is relevant in order to guarantee a high level of security. In this work we discuss a solution in order to address these possible execution contexts. Symmetric key primitives are considered as they generally correspond to the main bottleneck when dealing with high amount of data, they are also cheaper in terms of hardware resources. It leads to a better tradeoff in terms of additional cost and achieved performances.

### *B. Cryptoprocessor as a Trusted Computing Base*

The cryptoprocessor should provide algorithms for data privacy, data authentication, and hashing, furthermore a secure way to generate and store session keys is mandatory. Data privacy (i.e. confidentiality) guarantees the information can be understood only by authorized users, IV based encryption schemes are used to achieve this goal. NIST recommends Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), and Counter (CTR) [7]. In the context of symmetric key, data authentication means data integrity, i.e. it guarantees that the information has not been modified. Message Authentication Codes (MACs) correspond to the suitable cryptographic primitives to achieve this goal. In the literature there are several MAC schemes; interested readers can find a survey about them in [6]. In the context of manycore clusters, algorithms that provide a wide range of parallelism and high efficiency are expected. A selection of possible algorithms will be discussed latter in the paper.

Session keys should be generated using some of the existing True Random Number Generator [8]. The storage of session keys is performed inside the cryptoprocessor in a trust memory zone. As a security requirement when a key has to be transmitted outside the cryptoprocessor, the key is first encrypted and then sent to the GPP. The GPP has only access to encrypted keys in order to avoid software attacks like cache attacks [9]. To prevent some side-channel attacks, decryption primitive of block cipher should be used only to ensure the security of session keys. Other cryptographic services must only rely on the encryption primitive. Inside the cryptoprocessor an Arithmetic Logic Unit (ALU) is required. It performs the basic operations needed to compute mode of operations like bitwise XOR, increment, accumulators, etc.

From the system point of view the cryptoprocessor must be separated both at the architectural level and at the physical level from the GPP following the guidelines given in [10] which allow building a secure system by design.

### *C. Notations*

The next sections present the cryptographic parallel computation and the key exchange protocol. To describe these solutions, the following notation is considered: A block cipher is denoted as $E_K(\cdot)$ and n is its block length, the n-bit strings will be considered as members of $GF(2^n)$ or as polynomials of degree at most n-1. The addition in this field

is a bit-wise $\oplus$. + denotes an integer addition modulo $2^n$. A||B denotes the concatenation of A and B and |A| is the length in bits of A. Xtimes means the multiplication of polynomial $q(x)$ by monomial $x$ modulo an irreducible polynomial $p(x)$ and denoted as $xL$. In terms of values $xL = 2L$, $x^2L = 2L$ ,…, $x^mL = 2^mL$ this operation can be computed very efficiently using only one shift register and some XORs.

### *D. Performing cryptographic operations in parallel*

When the amount of data is large, for example when encrypting an image or authenticating a video, more than one cryptoprocessor can be used in parallel to reduce the computation time. Some existing cryptographic algorithms allow their implementation in parallel. For encryption, Counter mode is easy to parallelize, it is defined as follows:

$$C_1 \,||\, C_2 \,||\, \ldots \,||\, C_m =$$
$$E_K(IV) \oplus P_1 \,||\, E_K(IV+1) \oplus P_2 \,||\, \ldots \,||\, E_K(IV+m-1) \oplus P_m$$

where IV is an initialization vector, $P_1,\ldots,P_m$ the plaintext and $C_1,\ldots,C_m$ the ciphertext. We can see from the definition that each invocation of $E_K(\cdot)$ is totally independent from the previous, so that a message can be split into parts and each part can be sent to different cryptoprocessors along with the correct increment of the IV. For MAC the parallelization process is more complicated. As a first study Parallelizable Message Authentication Code (PMAC1) [11] is considered in this work, it is shown in Fig. 1:

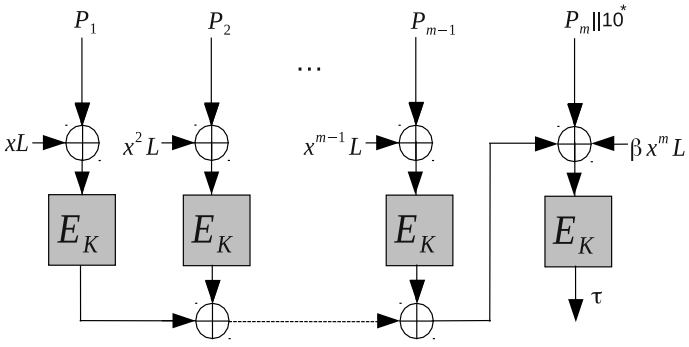


Fig. 1. PMAC1 for complete blocks, $L = xE_K(0^n)$, $\beta = 3$ if $P_m$ is a complete block otherwise $\beta = 5$.

Same as counter mode, in PMAC1 each invocation to $E_K(\cdot)$ is independent, but in the masking sequence $xL, x^2L,\ldots, x^mL$ each value is dependent from the previous, so the way to parallelized it is interleaving the blocks between the different cryptoprocessors. For example using two cryptoprocessors, one of them processes the odd blocks with the masking $xL, x^3L, x^5,\ldots$ and the other one the even blocks with the masking $x^2L, x^4L, x^6,\ldots$ .Each cryptoprocessor computes the intermediate addition and finally any of them computes the final tag $\tau$. The same principle can be considered when more than two cryptoprocessors are available. In the Table 1 we show the computational cost of PMAC and counter mode.

TABLE 1 COMPUTATIONAL COST OF PMAC AND COUNTER MODE

| *Cryptographic mode* | *BC* per block* | *XOR per block* | *Extra BC* | *Extra XOR* |
|---|---|---|---|---|
| *PMAC* | *1* | *1* | *0* | *0* |
| *Counter Mode* | *1* | *2* | *2* | *1* |

**BC is for block cipher calls*

When cryptoprocessors are used in parallel to perform some operation, all of them must use the same key, so a protocol for key exchange is necessary; this point is addressed in the following subsection.

### *E. Key exchange between cryptoprocessors*

To perform parallel computation using several cryptoprocessors, all of them must handle the data using the same key. The output of the cryptoprocessors (i.e. cleartext or ciphertext) should be the same independently of the number of cryptoprocessors used to perform the computation. Indeed depending of the load of the system, encryption (or decryption) may be performed using n cryptoprocessors while decryption (or encryption) may use m cryptoprocessors. This point is defined at runtime based on the number of applications running in parallel on the manycore architecture. Keys can be shared using asymmetric cryptography but the cost in hardware of implementing such cryptography is huge in comparison with symmetric key primitive. The disadvantage of symmetric key based protocols is that they need to preload some master keys in each cryptoprocessor, but they can be implemented using just a MAC and a TRNG.

In order to allow an efficient key exchange between cryptoprocessors we propose a three levels hierarchical organization of the keys. The first level is the master key provided from the initialization of the system, the second level is the session keys generated by the cryptoprocessor at runtime and finally the third level is a pair of keys used to share and generate session keys for collaborative work between a set of cryptoprocessors. The mentioned organization allows the isolation of cryptoprocessors into sets and the authentication of the elements of the sets. A protocol to share a session key based on Location-Based Pairwise Key Establishments for Static Sensor networks [1] can be used. First, all the elements of the set of the cryptoprocessors are preloaded with a group master key $K_G$ and set key generation key $K_K$, and a unique identifier ID.

The pairwise protocol between cryptoprocessor A and cryptoprocessor B is performed as follows:

1. A generates a nonce $N_A$ and sends it to B $[\text{Re}\,quest, A, N_A, MAC_{K_G}(Hello, A, N_A)]$.
2. B responds $[ACK, B, A, N_B, MAC_{K_G}(ACK, B, A, N_A, N_B)]$.
3. A can verify that B is a valid member of the group computing $MAC_{K_G}(ACK, A, B, N_A, N_B)$ and comparing with the value received from B.
4. The pairwise key is computed as $MAC_{K_K}(N_A, N_B)$.

Using this process the pairwise communication keys between cryptoprocessors are established, after this any cryptoprocessor can generate a session key and share it with other cryptoprocessors to perform cryptographic computation in a collaborative way.

If there is a secure channel between cryptoprocessors in a set, any of them can generate a session key using the TRNG and sends it to the other elements of the set using such a channel. The main drawback with this approach is the architectural implication of dedicating an exclusive channel for cryptoprocessors and its isolation from the other communication resources.

Just some operations are necessary to perform the agreement of a key between to cryptoprocessors. Only one call to TRNG to generate a nonce and three executions of underlying MAC are required: one to prepare the request, another one for authentication and the last one to generate the pairwise key. The computational cost of a MAC is measured according to the number of blocks. A possible scenario is to consider that ACK and Hello are 16-bit fixed values, the identifiers of each cryptoprocessor A and B are 32-bit values (enough $2^{32}$ different identifiers) and finally nonces $N_A$ and $N_B$ are 128-bit strings. If PMAC is used as the underlying MAC implemented with a 128-bit-block cipher like AES, the $|HELLO||A||N_A|$=176 bits means a two-block input to PMAC plus two additional block ciphers calls (see Table 1). The total cost of the MAC computation for request is four block ciphers calls. Following the same analysis, Table 2 shows the computational cost for all MACs involved in pairwise key agreement. If the amount of data to be processed is large, the cost of key agreement between a group of cryptoprocessors is negligible.

The key agreement is done only the first time that two cryptoprocessors collaborate for some task, after that the corresponding pairwise key is stored in a trust zone in each cryptoprocessor correctly indexed for its future use. The amount of storage is exactly one key for each cryptoprocessor in the group, and initially it is necessary to preload three keys in each cryptoprocessor: master key $K_M$, group key $K_G$ and generation key $K_K$.

Some of the existing MAC algorithms based on block ciphers are parallelizable but not at the level required to be easily computed in a cluster, so some work still needs to be done to explore new constructions of MACs and Authenticated Encryption modes with a wide level of parallelization and efficiency.

TABLE 2 COMPUTATIONAL COSTS OF MACS USED TO AGREE A PAIRWISE KEY

| *Operations involved in pairwise key agreement* | *Input-length* | *Block cipher calls* | *XOR* |
|---|---|---|---|
| $MAC_{KG}(Hello, A, N_A)$ | *176-bit* | *4* | *5* |
| $MAC_{KG}(ACK, B, A, N_A, N_B)$ | *304-bit* | *5* | *7* |
| $MAC_{KG}(N_A, N_B)$ | *256-bit* | *4* | *5* |
| ***Total for the petitioner*** | | ***13*** | ***17*** |
| ***Total for the responder*** | | ***14*** | ***19*** |

## III. CONTRIBUTIONS OF THE TSUNAMY PROJECT

The TSUNAMY project [12] explores some solutions to address main issues presented in Section II. The following section illustrates some of these solutions.

### *A. TSAR manycore originale architecture and enhancement*

The TSUNAMY project relies on the TSAR manycore architecture [13]. The TSAR architecture is based on clusters connected through a NoC. Each cluster contains some GPPs, some memories and peripherals.

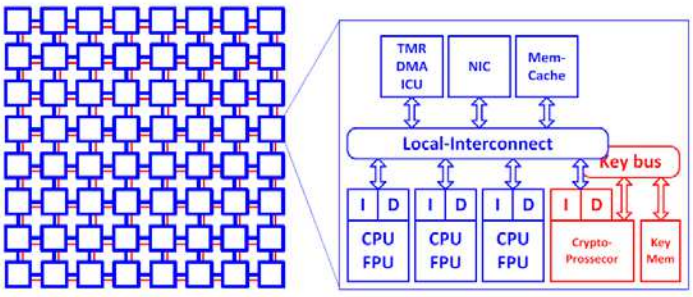


Fig. 2. Enhanced TSAR architecture

Fig.2 illustrates the TSAR architecture extended with a cryptoprocessor in each cluster. The cryptoprocessor is connected to the local interconnect and is reachable as any peripheral. DMA is used to send and retrieve data to/from the cryptoprocessor. This scheme allows reducing the load of CPUs. The cryptoprocessor used in this extended architecture is based on the HCrypt solution [10].

Cryptoprocessor has configuration registers and its own DMA controller. When the GPP wants to use the cryptoprocessor first it has to send commands to configuration registers to indicate the task to be performed by the cryptoprocessor, the initial memory address and the size of the data buffer for the DMA, the initial memory address where the cryptoprocessors will write the result of

its computation, also establish initial values if necessary. After all the configurations are ready, GPP sends and starts command to the cryptoprocessor. When the cryptoprocessor ends the task it sends a ready message to the GPP.

The following section presents the HCrypt cryptoprocessor used to the enhancement of TSAR.

### *B. HCrypt Architecture and implementation*

The HCrypt architecture is shown in Fig. 3. It is divided into three security zones: data, cipher and key zones [10]. The HCrypt data zone contains 128-bit data registers, 128-bit ALU, I/O FIFOs (conversion between 32 bits and 128 bits) and the control logic. Configuration data buses (in black) are located in the configuration data zone and partly in the cipher zone. The HCrypt cipher zone contains two ciphers that work independently. *Cipher$_1$* is dedicated to key protection and *Cipher$_2$* to data protection.

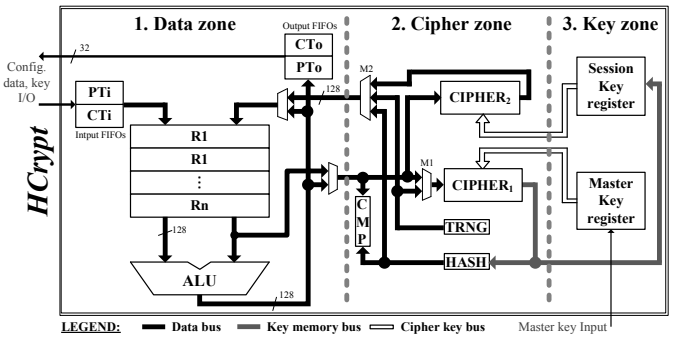


Fig. 3. Architecture of the HCrypt cryptoprocessor.

*Cipher$_1$* uses a master key to decrypt session keys. Random numbers generated by the TRNG are transferred directly as an encrypted session key which is decrypted on both sides using the cipher (*Cipher$_1$* in cryptoprocessor). For security reasons, *Cipher$_1$* cannot be used in CBC-MAC mode for authentication of the session key because its output would need to be connected to the external data bus (in the data zone) and the processor could access the session key in clear. Instead, we propose to use a hashing function for session key authentication. In this way, the session key cannot be read in clear from the session key register. The session key is transferred in clear only from *Cipher$_1$* to the key register and from the *Cipher$_1$* to the HASH function using the dedicated key memory bus (in gray). The master key and session key registers each stores only one key. Before encryption, encryption keys (master or session keys) are transferred to key inputs of both ciphers via dedicated cipher key buses (in white).

Data are deciphered only by *Cipher$_2$* using session keys. The cryptoprocessor supports all basic block encryption modes except ECB and CBC modes, because deciphering in both modes requires a decipher, which is not available in the HCrypt. However, data can be authenticated using the CBC-MAC mode, which requires only the cipher.

A prototype of HCrypt cryptoprocessor was described in VHDL and mapped to Virtex-6 XC6VLX240TFF1156 device using ISE ver. 12.4. It was implemented and tested in the Xilinx ML605 board. The proposed architecture uses only fine grain FPGA resources and embedded RAMs/FIFOs. HCrypt uses 1618 slices (including two AES ciphers, MD5 hash function, TRNG, data path including ALU, internal registers and control logic) and 1188 kbits of embedded RAM.

With a clock frequency of 100 MHz, the HCrypt cryptoprocessor reached the payload throughput of about 860 Mbits/s. To estimate this latency, Table 3 provides the latency in number of clock cycles for several cryptographic operations.

In Table 3 we can see the number of clock cycles taken by HCrypt to perform the counter mode and PMAC. Counter mode is very efficient because of the independence between the block cipher inputs, and the increment of the counter is performed in a dedicated register in the ALU of HCrypt. The Xtimes operations for PMAC is also computed inside a special register, the initial latency for PMAC is larger than in CBC-MAC because of the two extra block cipher calls. The extra latency of PMAC is not important when the amount of data to be processed is significantly large. In the case of key agreement process CBC-MAC could be used also in order to save some clock cycles.

The initial version of HCrypt is not designed to work in a collaborative way, so an important task of this work is to explore the way to use many cryptoprocessors together. We will explore the architectural and logical aspects to communicate cryptoprocessors in a secure way; this means the design of protocols that allow communicating of cryptoprocessors without the intervention of GPP. Another important task is the definition of the size of the groups of cryptoprocessors and the level of cooperation between the members, how many cryptoprocessors can be used for the same task and the limit of parallelization of the cryptographic algorithms.

TABLE 3 LATENCY OF HCRYPT OPERATION IN NUMBER OF CLOCK CYCLES

| *Cryptographic operations and modes* | *HCrypt hardware resource* | *Number of clock cycles* |
|---|---|---|
| *Session key generation* | *TRNG* | *1* |
| *128-bit AES ECB* | *Cipher$_1$* | *11* |
| *256-bit AES ECB* | *Cipher$_2$* | *22* |
| *MD5 (128-bit input data)* | *Hash* | *66* |
| *K.128-bit AES CFB* | *Cipher$_2$* | *14.K+5** |
| *K.128-bit CBC-MAC* | *Cipher$_2$* | *13.K+5** |
| *PMAC* | *Cipher$_2$* | *26*+13.K* |
| *Counter mode* | *Cipher$_2$* | *12.K* |

**5 clock cycles for initialization vector, 26 clock cycles for the two extra block ciphers calls and extra XOR.*

## IV. Discussion

Building a trusted solution based on manycore architecture still requires some work in order to build a trust chain between the applications and the architectures. The software layer is a critical point where most of attacks are launched. It is thus essential to define the software Trusted Computing Base. Different solutions can be considered based on an operating system or combining an operating system and a virtualization layer. One key point is application isolation in order to guarantee a secure execution of application and no leakage of information (code, data). Some solutions to this issue propose to use virtualization mechanism [14], to use OS-level separation for multiprocessor system on chip [15], or recently to run a trusted agent (TCB element) on every core of a many-core platform [16].

Combining software and hardware isolation needs to be considered in order to build a strong security layer being able to protect the system against confidentiality, integrity and denial of services attacks. It is also essential to extensively analyze the threat model in order to anticipate the risks when the application or/and the kernel are compromised. There have been many researches in that domain, it is required to adapt them to the manycore context where many applications can run concurrently and compete for some resources. Another important concern is how to distribute applications on the platform, how to schedule the applications and how to build some efficient heuristics taking into account processor workloads, execution parameters (temperature, frequency, and voltage) and security. Introducing security as a new dimension is a major concern and still many works needs to be done in that direction.

In the TSUNAMY project we address some of these challenges and our goal is to validate these propositions through simulation modeling using SystemC CABA. For that purpose the TSAR architecture and the ALMOS operating system [17] are used. The TSUNAMY project aims to provide the scientific community of academic and industrial with models of architectures and software libraries to efficiently and securely deploy applications on manycore architectures.

## V. Conclusion

Manycore architectures correspond to an important computation paradigm shift for modern embedded systems. The secure execution of applications using these architectures still needs to be investigated. In this paper we propose to stress some of the challenges that are in front of us and we discuss some possible solutions in order to enhance these architectures with cryptoprocessors. Key agreement in a main step when several cryptoprocessors are used in parallel to increase the computation efficiency. We propose a first approach to address this point and evaluate the performance impact. Some research directions are also discussed and conclude the paper.


## Acknowledgment

The work presented in this paper was realized in the frame of the TSUNAMY project number ANR-13-INSE-0002-02 supported by the French ”Agence Nationale de la Recherche”.


## Biblography


[1] Francisco Rodríguez-Henríquez, N. A. Saqib, A. Díaz-Pèrez, and Cetin Kaya Koc. 2006. Cryptographic Algorithms on Reconfigurable Hardware (Signals and Communication Technology). Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[2] Ted Krovetz and Phillip Rogaway. 2011. The software performance of authenticated-encryption modes. In Proceedings of the 18th international conference on Fast software encryption (FSE'11), Antoine Joux (Ed.). Springer-Verlag, Berlin, Heidelberg, 306-327.

[3] Chakraborty, D., Mancillas-Lopez, C., Rodriguez-Henriquez, F., & Sarkar, P. (2013). Efficient hardware implementations of brw polynomials and tweakable enciphering schemes. Computers, IEEE Transactions on, 62(2), 279-294.

[4] Fengguang Song and Jack Dongarra. 2012. A scalable framework for heterogeneous GPU-based clusters. In Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures (SPAA '12). ACM, New York, NY, USA, 91-100.

[5] Morris J. Dworkin. 2005. SP 800-38b. Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. Technical Report. NIST, Gaithersburg, MD, United States.

[6] Marcos A. Simplicio, Jr, Bruno T. De Oliveira, Cintia B. Margi, Paulo S. L. M. Barreto, Tereza C. M. B. Carvalho, and Mats NäSlund. 2013. Survey Survey and comparison of message authentication solutions on wireless sensor networks. Ad Hoc Netw. 11, 3 (May 2013), 1221-1236.

[7] Morris J. Dworkin. 2001. SP 800-38A 2001 Edition. Recommendation for Block Cipher Modes of Operation: Methods and Techniques. Technical Report. NIST, Gaithersburg, MD, United States.

[8] Viktor Fischer, Florent Bernard. 2011. True Random Number Generators in FPGAs. Security Trends for FPGAS. Benoit Badrignans, Jean Luc Danger, Viktor Fischer, Guy Gogniat and Lionel Torres (Eds.). Springer Netherlands.

[9] Eran Tromer, Dag Arne Osvik, and Adi Shamir. 2010. Efficient Cache Attacks on AES, and Countermeasures. J. Cryptol. 23, 2 (January 2010), 37-71.

[10] Lubos Gaspar, Viktor Fischer, Lilian Bossuet, and Robert Fouquet. 2012. Secure Extension of FPGA General Purpose Processors for Symmetric Key Cryptography with Partial Reconfiguration Capabilities. ACM Trans. Reconfigurable Technol. Syst. 5, 3, Article 16 (October 2012), 13 pages.

[11] P. Rogaway. 2004. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC, in: Advances in Cryptology –Asiacrypt'04, Lecture Notes in Computer Science, vol. 3329,Springer-Verlag, Heidelberg, Germany, pp. 16–31.

[12] The TSUNAMY project www.tsunamy.fr

[13] The TSAR manycore architecture https://www-soc.lip6.fr/trac/tsar

[14] M. Pearce, S. Zeadally, and R. Hunt. 2013. Virtualization: Issues, security threats, and solutions. ACM Comput. Surv. 45, 2, Article 17 (March 2013), 39 pages.

[15] H. Inoue, J. Sakai, S. Torii, and M. Edahiro. 2009. FIDES: An advanced chip multiprocessor platform for secure next generation mobile terminals. ACM Trans. Embed. Comput. Syst. 8, 1, Article 1 (January 2009), 16 pages.

[16] R. J. Masti, D. Rai, C. Marforio, and S. Capkun. 2014. Isolated execution in many-core architectures. Network and Distributed System Security Symposium.

[17] The ALMOS operating system https://www-soc.lip6.fr/trac/almos